Experience report: Mechanizing Core F$^{\forall}$ using the locally nameless approach

(extended abstract)

Benoît Montagu
INRIA Paris-Rocquencourt
benoit.montagu@inria.fr

Abstract

For a couple of years, much effort has been put in the development of techniques that ease the mechanization of proofs involving binders. We report such a mechanized development of metatheory, the type soundness of Core F$^{\forall}$ [3], by a non expert user of Coq [2], using the locally nameless representation of binders and cofinite quantification, with the help of the tools LNgen [1] and Ott [4].

1. F$^{\forall}$ and its formal proof in a nutshell

Core F$^{\forall}$ (F-zip) is a variant of System F that allows for more freedom in the structure of programs that make use of existential types, by considering existentials with an *open* scope. It is equipped with a small-step reduction semantics and a sound type system.

The paper proof is neither very informative, nor very difficult, and consists in the *subject reduction* and the *progress* properties. The mechanized proof was carried out in about one month by the author, who is not an expert user of Coq. It makes use of LNgen [1] and the experimental locally nameless backend of Ott [4] to reduce the burden of the locally nameless encoding and its infrastructure lemmas. The only complex automation we used is the one provided by the Metatheory library from UPenn, that was of great help. Thus, a clean up of the development as well as clever tactics could certainly reduce the size of the whole proof. Much time is spent in proof search, so that Coq compiles it in about 45 minutes on a recent computer, while type checking takes just a few minutes.

Specifications	Proofs	
3 000	1 450	Automatically generated
2 600	8 600	Manually produced
5 600	10 050	Total

Specification/proof ratio: 35%; Ratio produced by tools: 28%.
http://gallium.inria.fr/~montagu/proofs/FzipCore/

Table 1. Size of the development in lines of code.

2. Encountered issues

In spite of its relatively simple paper proof, Core F$^{\forall}$ gathers a set of peculiarities, that makes it not straightforward to mechanically formalize:

- The syntax of terms is not stable under substitution of types, which forced us to slightly change the definition of the system;
- Reduction proceeds under some binders, which makes renaming lemmas on the reduction relation necessary, for example, to prove the *progress* property;
- Some reduction rules involve nested binders, while some other ones *create* new binders in their conclusion: this is the case of rules that extrude the scope of a binder, or that *swap* two consecutive binders. This makes the encoding using cofinite quantification somewhat convoluted, or nay difficult to understand, so that we had to prove the admissibility of more natural formulations of the rules just to convince ourselves that the encoding was correct. Ott currently failed to handle these rules in a correct manner.

We now quickly illustrate each of the above points, as well as the lack of support for definitions of functions over terms with bindings.

2.1 A syntax that is not closed under substitution

The syntax of terms in Core F$^{\forall}$ is not stable under substitution of types. Two constructs are responsible for this: open $\langle\alpha\rangle\ M$ and $\Sigma\ \langle\alpha\rangle\ (\beta = \tau)\ M$, where the free type variable α can only be replaced with another type variable. LNgen however only has support for syntax that are closed under substitution. As a consequence, we were forced to extend our definition (*i.e.* allow open $\langle\tau\rangle\ M$ and $\Sigma\ \langle\tau\rangle\ (\alpha = \tau')\ M$ for arbitrary τs), and then restrict the uses of the syntax in the typing rules.

We think that *renaming* should be considered as the basic operation on terms with binders, instead of *substitution*.

2.2 Definition of functions over structures with binders

We sometimes needed to define functions over terms. LNgen provides a recursion combinator on locally closed terms for this purpose. For every binder to be opened, we need to compute a sufficiently fresh atom. This is problematic when one reasons about such functions, as one often needs to use another atom than the one that was chosen to open a binder. We consequently preferred to define them as relations using the cofinite quantification and then prove they indeed define functions. This necessitated renaming lemmas to prove the existence of an image for every argument.

2.3 Reduction under binders

In Core F$^{\forall}$, reduction proceeds under some constructs that bind type variables. As a consequence, the contextual closure of the reduction relation uses cofinite quantification when traversing binders. This is automatically done by Ott's experimental locally nameless backend.

For instance the reduction rule that says $\nu\alpha.\, M \rightsquigarrow \nu\alpha.\, M'$ provided $M \rightsquigarrow M'$ is expressed in Coq as follows, where $\mathcal{L}$ denotes a finite set of atoms:

$$\frac{\forall \alpha \notin \mathcal{L},\ \texttt{open\_term}\ M\ \alpha \rightsquigarrow \texttt{open\_term}\ M'\ \alpha}{\nu M \rightsquigarrow \nu M'}$$

But it is often the case (as in the proof of *progress*) that one wants to conclude while the premise is true only for *some* atom α, hence the need for the rule:

$$\frac{M \rightsquigarrow M'}{\nu\,(\texttt{close\_term}\,\alpha\,M) \rightsquigarrow \nu\,\big(\texttt{close\_term}\,\alpha\,M'\big)}$$

It is well known that the proof of validity of this rule is not direct: it requires a renaming lemma.

2.4 Nested binders

Let us now consider the following reduction rule: $\nu\beta.\,\Sigma\,\langle\beta\rangle\,(\alpha = \tau)\,r \rightsquigarrow r\,[\beta \leftarrow \alpha]\,[\alpha \leftarrow \tau]$, provided $\beta\sharp\tau$ and r satisfies the `result` predicate. In this rule, the ν construct binds β, while α is bound in r by the Σ construct. The problem is that the cofinite quantification method makes this rule much less readable:

$$\frac{\begin{array}{l}\forall\beta \notin \mathcal{L}, \forall\alpha \notin \mathcal{L} \cup \{\beta\},\\ \quad \texttt{open\_term}\,(\Sigma\,0\,\tau\,M)\,\beta = \Sigma\,\beta\,(\texttt{open\_typ}\,\tau\,\beta)\,M_1 \rightarrow\\ \quad \texttt{open\_term}\,M_1\,\alpha = M_2 \rightarrow\\ \qquad\qquad \beta \notin \texttt{open\_typ}\,\tau\,\beta \;\wedge\; \texttt{result}\,M_2\\ \qquad\quad \wedge\; M' = M_2\,[\beta \leftarrow \alpha]\,[\alpha \leftarrow \texttt{open\_type}\,\tau\,\beta]\end{array}}{\nu\,(\Sigma\,0\,\tau\,M) \rightsquigarrow M'}$$

Notice that we need to consider M under all its bindings, hence the introduction of M_1 and M_2. It is here rather easy to prove that the relation really has an image, *i.e.* M' is well defined, since $M_2 = \texttt{open\_term}\,\big(\texttt{open\_term}^1\,M\,\beta\big)\,\alpha$ (the 1 in superscript refers to the De Bruijn indice 1; we simply omit it if it is 0).

We could not directly define M' in the conclusion, since it would use atoms that are cofinitely bound in the hypothesis. As the encoding is not obvious, we proved the admissibility of the rule:

$$\frac{\texttt{result}\,r \qquad \beta \sharp \tau}{\begin{array}{l}\nu\,(\texttt{close\_term}\,\beta\,(\Sigma\,\beta\,\tau\,(\texttt{close\_term}\,\alpha\,M))) \rightsquigarrow\\ \qquad\qquad M\,[\beta \leftarrow \alpha]\,[\alpha \leftarrow \tau]\end{array}}$$

This kind of rule, that we name "in closed form" as they use `close` in their conclusion and avoid the use of `open`, closely resembles the original definition, as the occurrences of binders are indicated by occurrences of `close`.

2.5 Extrusion of binders

Another set of reduction rules implement the *extrusion* of the Σ construct, for instance through pairs: $(\Sigma\,\langle\beta\rangle\,(\alpha = \tau)\,r_1, r_2) \rightsquigarrow \Sigma\,\langle\beta\rangle\,(\alpha = \tau)\,(r_1, r_2\,[\beta \leftarrow \alpha])$ provided $\alpha \sharp r_2$ and r_1 and r_2 are results. As previously, the created binder in the result cannot be completely defined in the conclusion, since it would use cofinitely quantified atoms. We had to encode this rule as follows:

$$\frac{\begin{array}{c}\texttt{result}\;(\Sigma\,\beta\,\tau\,M_1) \qquad \texttt{result}\,M_2\\ \forall\alpha \notin \mathcal{L},\; \texttt{open\_term}\,M_2'\,\alpha = M_2\,[\alpha \leftarrow \beta]\end{array}}{(\Sigma\,\beta\,\tau\,M_1, M_2) \rightsquigarrow \Sigma\,\beta\,\tau\,\big(M_1, M_2'\big)}$$

where the term M_2' in the conclusion is specified in the premise by considering its opened version. Again, we proved the validity of the following rule in closed form, to convince ourselves that the encoding was correct.

$$\frac{\texttt{result}\,M_1 \qquad \texttt{result}\,M_2 \qquad \alpha \sharp M_2}{\begin{array}{l}(\Sigma\,\beta\,\tau\;(\texttt{close\_term}\,\alpha\,M_1)\,, M_2) \rightsquigarrow\\ \qquad\qquad \Sigma\,\beta\,\tau\;(\texttt{close\_term}\,\alpha\;(M_1, M_2\,[\beta \leftarrow \alpha]))\end{array}}$$

2.6 Swapping of binders

The situation is worse when considering operations that *swap* binders, as specified by the following reduction rule: $\Sigma\,\langle\beta_1\rangle\,(\alpha_1 = \tau_1)\,\Sigma\,\langle\beta_2\rangle\,(\alpha_2 = \tau_2)\,r \rightsquigarrow \Sigma\,\langle\beta_2\rangle\,(\alpha_2 = \tau_2\,[\alpha_1 \leftarrow \tau_1])\,\Sigma\,\langle\beta_1\rangle\,(\alpha_1 = \tau_1)\,r$, provided r is a result and $\alpha_2 \sharp \tau_1$. We expressed this rule with cofinite quantification as follows:

$$\frac{\begin{array}{l}\forall\alpha_1 \notin \mathcal{L}, \forall\alpha_2 \notin \mathcal{L} \cup \{\alpha_1\},\\ \quad \texttt{result}\,\big(\texttt{open\_term}\,\big(\texttt{open\_term}^1\,M\;\alpha_2\big)\,\alpha_1\big) \wedge\\ \quad \tau_1 = \texttt{open\_typ}\;\tau_1'\,\alpha_2 \wedge\\ \quad \texttt{open\_term}\,\big(\texttt{open\_term}^1\,M\,\alpha_1\big)\,\alpha_2\\ \qquad\qquad = \texttt{open\_term}\,\big(\texttt{open\_term}^1\,M'\,\alpha_2\big)\,\alpha_1\end{array}}{\Sigma\,\beta_1\,\tau_1(\Sigma\,\beta_2\,\tau_2\,M) \rightsquigarrow \Sigma\,\beta_2\,(\texttt{open\_typ}\;\tau_2\,\tau_1)\,\big(\Sigma\,\beta_1\,\tau_1'\,M'\big)}$$

As in the previous examples, we had to consider the term M under its binders, hence we have to open it twice (once at the De Bruijn level 1, as specified in superscript, and another time at level 0) to implement the swapping of binders. But since it is very easy to get the levels or the order of openings wrong, we were only convinced by the correctness of our encoding, once we proved the validity of the rule in closed form, that mimics the definition on paper:

$$\frac{\texttt{result}\,r \qquad \alpha_2 \sharp \tau_1}{\begin{array}{l}\Sigma\,\beta_1\,\tau_1\,(\texttt{close\_term}\,\alpha_1\,(\Sigma\,\beta_2\,\tau_2\,(\texttt{close\_term}\,\alpha_2\,r))) \rightsquigarrow\\ \Sigma\beta_2\,(\tau_2\,[\alpha_1 \leftarrow \tau_1])\,(\texttt{close\_term}\,\alpha_2\,(\Sigma\beta_1\,\tau_1\,(\texttt{close\_term}\,\alpha_1\,r)))\end{array}}$$

The proof of validity not only required a renaming lemma, but also a specific one about indices, that was hard to state well:

Lemma 2.1. *For any α, α_1, α_2, M and n,*

$$\begin{array}{l}\texttt{open\_term}^n\,\alpha_1\,\big(\texttt{open\_term}^{n+1}\,\alpha_2\,\big(\texttt{close\_term}^{n+1}\,\alpha\,M\big)\big)\\ = \texttt{open\_term}^n\,\alpha_2\,\big(\texttt{open\_term}^{n+1}\,\alpha_1\,(\texttt{close\_term}^n\,\alpha\,M)\big)\end{array}$$

It was an unpleasant surprise, that we had to work with the implementation of binding: the user should not be confronted to it. We must admit that we tried several versions of this lemma, until we eventually got it right by chance.

3. Conclusive remarks

Although it provides powerful induction principles, cofinite quantification is not the *panacea* when it is used as a specification: it can render your judgments hard to understand, and the encoding is sometimes not obvious and error prone. Moreover, we sometimes had to deal with De Bruijn indices, which is unsatisfactory.

Instead, we think that rules in closed forms are easier to understand and hence should be preferred in specifications. But this would require some support to derive strong induction principles, as done in Nominal Isabelle [5]. Rules in closed forms also naturally appear when one requires that each quantified term in a rule is also locally closed. Moreover, we think that using closed forms would simplify the work for Ott and extend its range of applicability.

To our limited knowledge, defining and reasoning on functions over terms with bindings in Coq is still an annoying problem, for which tool support would be much appreciated.

Although dealing with binders with Coq has improved a lot over the past years, some efforts are still needed, so that we can claim that mechanized metatheory is accessible to the masses.

References

[1] Brian Aydemir and Stephanie Weirich. LNgen: Tool support for locally nameless representations. Draft.

[2] The Coq development team. *Reference manual of the Coq proof assistant*, version 8.2 edition.

[3] Benoît Montagu and Didier Rémy. Modeling abstract types in modules with open existential types. In *Proceedings of ACM SIGPLAN Symposium on Principles of Programming Languages*, January 2009.

[4] Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, and Rok Strnivsa. Ott: Effective tool support for the working semanticist. *JFP*, 20(1):71–122, 2010.

[5] Christian Urban. Nominal reasoning techniques in Isabelle/HOL. *Journal of Automatic Reasoning*, 40(4):327–356, 2008.